Characterizing the Performance of Applied Intelligent Agents in Soar

Randolph M. Jones
Steve Furtwangler
Mike van Lent
SoarTech
3600 Green Court, Suite 600
Ann Arbor, MI 48105
734-327-8000
rjones@soartech.com, furtwangler@soartech.com, vanlent@soartech.com

Keywords:
Applied intelligent agents, Soar, Cognitive architecture, Performance evaluation

ABSTRACT: *There have been a few efforts to evaluate the robustness of performance of the Soar cognitive architecture, with positive results. However, previous efforts have focused primarily on running the architecture with agent models that are research systems, designed specifically for Soar performance evaluation, or otherwise limited in capability. This paper reports an effort to take a number of applied intelligent agents "off the shelf" and use them for a further evaluation of Soar. One primary goal is to see whether the performance results for research systems also hold for applied agents. A second goal is to characterize Soar's "practical" ability to run applied, knowledge-rich agents with performance that will scale in number of agents, memory requirements, and execution time.*

1. Introduction

We have a longstanding interest in the deployment of knowledge-rich intelligent agent systems in operational environments, particularly for DoD applications. For the practical application of these agent systems, we require a platform that is stable, robust, and operates within acceptable memory and execution-time constraints. A variety of efforts have characterized the Soar cognitive architecture (Laird, Rosenbloom, & Newell, 1987; Newell, 1990) and its ability to serve as the engine for such intelligent agents. However, these studies have largely focused on agent systems that are not applied or deployed systems, and/or agent systems that were developed primarily with the evaluation of the Soar architecture in mind. For this study, we consider *applied agents* to be interactive intelligent systems that were built to generate a particular set of target behaviors for some application, within the time and budget constraints of one or more projects. In addition, we consider applied agents in general (with some exceptions) to incorporate more complexity in terms of knowledge and representation than systems built specifically for research or performance evaluation.

To respond to this situation, we have run a number of additional performance tests on Soar, but using agent systems that were developed for DoD projects in which the evaluation of Soar was not a goal. These agent systems were developed with the primary goal of meeting application-specific reasoning and behavior generation requirements. Efforts to optimize the efficiency of these systems were of secondary emphasis, as long as the systems met the project requirements. We have several goals for this study:

- To determine whether the Soar performance results demonstrated by other studies also carry over to applied agent systems
- To determine whether the performance of applied Soar agents will degrade over the course of long-running tasks
- To determine how much these applied agents can be optimized beyond what was required to satisfy the project requirements and what these additional optimizations consist of
- To determine the number of applied intelligent agents that can be expected to run within acceptable performance constraints on typical, modern computer hardware

2. The Soar Cognitive Architecture

Soar is a software architecture for building computational cognitive models and intelligent agent systems. Soar's main execution loop is called the *decision cycle*, which consists of a fixed sequence of phases, shown in Figure 1. In the input phase, input from an external environment becomes represented as a set of interconnected symbols in a short-term *Working Memory*. In the elaboration phase all relevant long-term knowledge patterns *trigger and fire* to elaborate the system's representation of its current situation and to propose *operators* (discrete, deliberate actions) that are applicable in that situation. The knowledge patterns are implemented as *productions*, which are abstract patterns with conditions and actions, somewhat akin to logical implication rules.

Multiple candidate operators are compared based on preference knowledge, resulting in the selection of a single operator for execution per decision cycle. The selected operator executes by firing associated productions that specify the actions to be performed. At

the end of the decision cycle, Soar generates new outputs to enact actions in the external environment.

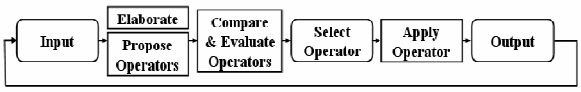

Figure 1. Phases of the Soar decision cycle.

In addition to being a software architecture in itself, Soar represents the implementation of a cognitive theory of the mind. In the typical application of the theory, Soar decision cycles map onto the types of deliberate, mental actions that occur in the human brain about once every 50 milliseconds (Newell, 1990). Although Soar developers do not always adhere directly to the theoretical constraints, the 50 msec/decision value is a target that is often aimed for, if only in system design, if not in actual implementation.

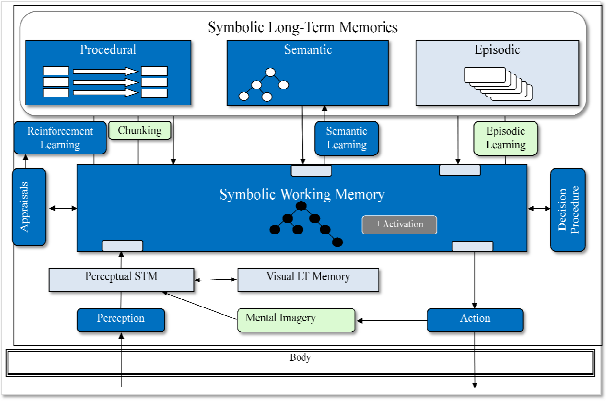

Figure 2. Soar 9 architecture, including reinforcement learning, semantic memory, and episodic memory.

3. Previous Evaluations

Through a variety of research and experimentation efforts, Soar has been demonstrated to operate effectively and efficiently using traditional computer architectures on large knowledge bases, and to perform robustly when running for long execution times. Doorenbos (1995) demonstrated that the contemporary implementation of Soar supported systems with over 100,000 productions in long-term memory with no degradation in execution time. More recently, Laird (2009) demonstrated that some fairly simple agents (in terms of capability, but not in terms of knowledge base size) could acquire over 17 million rules and run for over 10 million decision cycles, again with no degradation in performance. Laird also ran agents with smaller numbers of rules for billions of decision cycles, again with no degradation in performance. Additionally, Laird, Derbinsky, and Voigt (2011) have demonstrated robust performance results with systems that acquire large amounts of new knowledge using the declarative (semantic, and episodic) memory and learning modules introduced in Version 9 of the Soar architecture (see Figure 2). Our goal is to supplement these existing studies with further studies of the performance of Soar using applied agent systems that were built for capability and not primarily to measure Soar's robustness or speed. A significant question for our investigation is whether the assumptions and performance results that held for the research systems in the prior evaluations will also hold for a variety of applied systems. For such applied systems, we have various anecdotal data. For example the TacAir-Soar system (Jones et al., 1999) in 1997 contained approximately 8,000 productions and was able to perform tactical air combat missions in real time for up to 8 continuous hours of operation. For Soar work in general, the standard target for "real time" has been 50 milliseconds per decision cycle (Newell, 1990). However, this paper will provide more concrete data on the performance and robustness of such applied systems, include cycles times that are at least an order of magnitude faster that 50 milliseconds.

In the following section we describe each of the applied agent systems examined in the current study. Table 1 presents a quantitative summary of the complexity of these agents via two common measures, as well as the agents used in the study by Laird et al. (2011).

Table 1. Comparison of agent complexity. Working Memory (WM) size reflects the complexity of the agent's internal state representation. # of productions reflects the complexity of the agent's long-term knowledge.

Agent	# of productions	Avg WM size
Research Systems		
Simple robot	22	~250
Complex robot	530	~3000
Applied Systems		
Comm	224	~1600
JTAC	274	~7600
RWA-CAS	2045	~10500
RWA-DAS	2045	~6500
RWA-SAR	2045	~9000

4. Applied Agents

For the purposes of this study, we selected a number of existing applied Soar agents, developed by SoarTech under a combination of internal and external projects. One of the criteria for our selection of agents was that they all operate within a single interactive environment. Thus, all of the selected agents have been integrated into SoarTech's SimJr software simulation environment (Taylor & Ray, 2008), which provides low-cost simulation of entities and terrain for a variety of DoD-relevant missions and applications. This is partly to

reduce the chance that different execution environments will play a significant role in our measurements. Although we are also interested in assessing general "practical" performance of these agents, we want to be able to control the environment to some extent. That said, each of the agents performs a different type of mission, with different sensing, understanding, and reasoning requirements. So we expect to see at least some variation across agents, in at least some of the performance measures. Following, we describe each of the agents used in our experiments:

4.1 Wait Task Agent

The Wait Task agent can be considered the simplest Soar agent that is actually a complete Soar program. It consist of a single production that essentially "makes the decision" to do nothing during each of Soar's decision cycles. Thus, this agent provides a comparison baseline for the other agents, in terms of execution time and memory resources. We ran the Wait Task agent for an hour of real (wall clock) time, collecting statistics every 10 seconds.

4.2 Comm Agent

The Comm agent is one of a set of agents developed by SoarTech to simulate the communications patterns between various commanders in a Navy battle fleet. The current command scenario focuses heavily on communication and very little on any kind of sophisticated reasoning. Thus, we primarily expect to see memory and interaction effects for the Comm agent, as opposed to intensive computation for decision making. There are five Comm agents that send messages to each other, but each has a similar performance profile. Thus, our results include only the data from one of the five agents (the one that engages in the most communication during the command scenario). The Comm agent includes 224 productions in its long-term memory. The command scenario takes about 10 minutes of real time to run, so we collected performance statistics every 10 seconds for 10 minutes.

4.3 JTAC Agent

The JTAC agent performs the Joint Terminal Air Controller mission in simulations of Close Air Support (CAS). CAS missions involve a spotter (usually on the ground) who calls target information to an aircraft who delivers weapons against the target. CAS missions involve a high degree of communication and coordination, as well as risk assessment, because the targets are in close proximity to friendly forces. The JTAC agent for this experiment identifies targets visually (in simulation) and engages in a mission briefing, correction, monitoring, and execution process with an airborne weapons platform. In this scenario, the weapon platform is a simulated rotary-wing aircraft (RWA, described in the following section). The JTAC agent includes 174 productions in its long-term memory. For the experimentation scenario, we equipped the RWA assets with an unrealistic number of missiles so that the JTAC can continue running repeated CAS missions against targets for a full hour. We ran the scenario for an hour of real time, collecting performance statistics every 10 seconds.

4.4 RWA Agent

For the purposes of this experiment, the RWA agent serves as three different agents. That is, we used a single RWA agent model, consisting of 2045 productions in long-term memory. However, we measured the agent's performance in three different mission areas and scenarios. This allows us to characterize agent performance that is more a function of its tasking than its knowledge base.

In the RWA-CAS scenario, the RWA agent provides the weapons support for the JTAC agent described above. For our experiment, we placed an unrealistically large number of CAS targets and gave the RWA vehicle an unrealistically large number of simulated hellfire missiles. This allowed us to run the scenario with fairly continuous reasoning and activity for an hour of real time, collecting performance statistics every 10 seconds.

In the RWA-DAS scenario, the same RWA agent performs a Direct Air Strike mission against a number of targets. This mission involves the coordination of four different aircraft (each controlled by an instance of the RWA agent code) to plan attack routes, follow attack routes, then coordinate attacks against the targets. We again ran the scenario for an hour of real time, collecting performance statistics every 10 seconds. In this scenario, there is one Lead aircraft and three support aircraft. The performance profiles of the support aircraft are similar but somewhat less demanding than for the Lead aircraft (because the Lead does most of the decision making). Thus, in this paper we are only reporting the statistics collected for the Lead RWA agent.

In the RWA-SAR scenario, the same RWA agent performs a search-and-rescue mission, which mostly consists of a systematic search of an area and reporting of observations. An important feature that distinguishes this scenario from the other RWA scenarios is that the agent is configured to report a large amount of its internal reasoning to an external system (called VISTA) that provides graphical visualization of the agent's "mental state" (Taylor et al., 2002). This imposes additional performance, memory, and interaction requirements on the agent. Once again, we ran the RWA-SAR scenario for an hour of real time, collecting performance statistics every 10 seconds. For each of the RWA scenarios, we

ran experiments using the original RWA code base, as well as experiments after a set of fairly routine optimizations to the code (see the section below on Opportunities for Improvement). For the initial performance results we show only the optimized RWA performance. Performance of the RWA agent before optimization had the same characteristic shape with slightly worse performance.

5. Experimental Results

The following sections present the primary results and analysis of the collection of experiments described above. The goals of our analysis are to determine how the memory and computational performance of the agents scales over time, as well as assess the "practical" scalability of applied agent systems. We will describe what we mean by "practical" in the relevant subsection.[1]

5.1 Scalability of Memory Requirements

One of the results demonstrated by prior efforts to evaluate Soar is that memory demands do not continue to increase in an unbounded fashion with prolonged execution times (Laird, 2009; Laird, Derbinsky, & Voigt, 2011). This is not necessarily an obvious result. Certainly it is possible to build Soar agents that increasingly consume memory (by continually increasing the size of Working Memory) over time. And it is in fact the case that such agents will eventually slow to a crawl or exhibit other undesired characeteristics. But it is also the case that intelligent reasoning systems must base their decisions on some representation of situational understanding. Situational understanding does not just mean understanding what is going on in the current snapshot of time. It also means maintaining a memory of past events that have relevance on the interpretation and understanding of future events. Because of this need to base understanding in part on persistent memories, it is reasonable to assume that an applied intelligent agent *might* exhibit undesired memory growth. Our first analysis aims at discovering whether these applied agents (which again were optimized for capability, not necessarily for memory use) show any continual growth in memory use over time.

[1] The experiments used Soar 8.6.3 on an HP EliteBook with a 2.56 GHz Intel Core 2 Processor, and 3 GB of RAM, with Microsoft Windows XP Service Pack 3. Soar 8.6.3 was the version integrated into the SimJr environment. Soar 9 integrates several new learning and memory mechanisms (Laird, 2008), but none of the applied agents in this study use those mechanisms. A series of baseline comparisons verified that performance differences for these agents are not significant between Soar 8 and Soar 9.

Figure 3 displays the size of Soar's Working Memory for each agent during each 10-second snapshot. As expected, the Wait Task agent makes negligible demands on memory. The Comm agent also makes modest demands on Working Memory, because it engages mostly in communication, and not significantly on the types of decision making that require sophisticated situation understanding. The other agents make higher demands on memory, but the important point to note is that, for these applied agents, we see a consistent pattern of reaching an asymptote in Working Memory size that persists for the duration of execution. Thus, for relatively sophisticated and capable agents, our results replicate prior studies; we do not see an undesired pattern of memory growth. It is also interesting to note that the asymptote is different for each of the RWA agent missions, reiterating that representations of short-term knowledge are sensitive not only to the structure and capabilities of the agent, but also to the types of missions and situations assigned to the agent. The lesson here is that building capable applied agents does not require unmanageable growth in the internal representations those agents build and use.

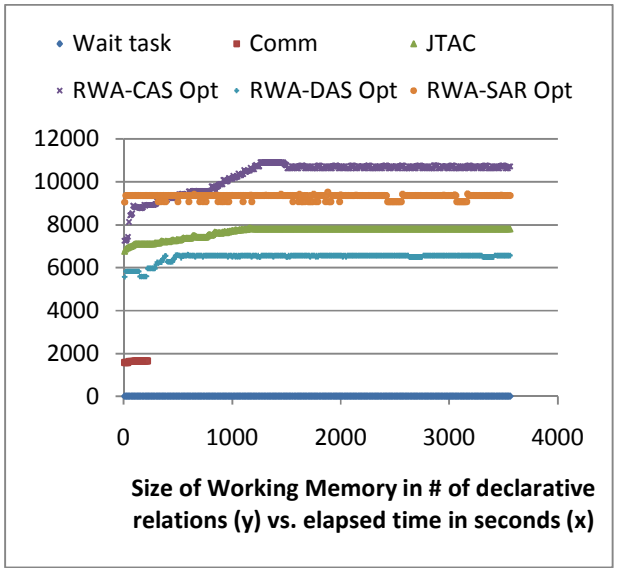

Figure 3. Change in Working Memory requirements over time.

5.2 Scalability of Execution Time

In addition to memory use, perhaps the most important factor contributing to scaleability of intelligent agents is response time. It is certainly reasonable to expect intelligent systems to have fairly expensive computational requirements, because they generally have to do sophisticated understanding and decision making. In highly dynamic environments (such as most of the environments in our experiments) these potentially expensive computations must occur frequently. An important question is whether the execution performance of an agent will degrade as we increase the capabilities of

the agent or during the "execution lifetime" of the agent. Doorenbos (1995) and Laird (2009) have demonstrated that even agents that have millions of productions in long-term memory, as well as agents that run for extremely long times, do not degrade in reaction time. However, it could be argued that these prior results were obtained from research agents. Thus, part of our intention with this investigation was to see if we see similar execution-time patterns in applied agent systems.

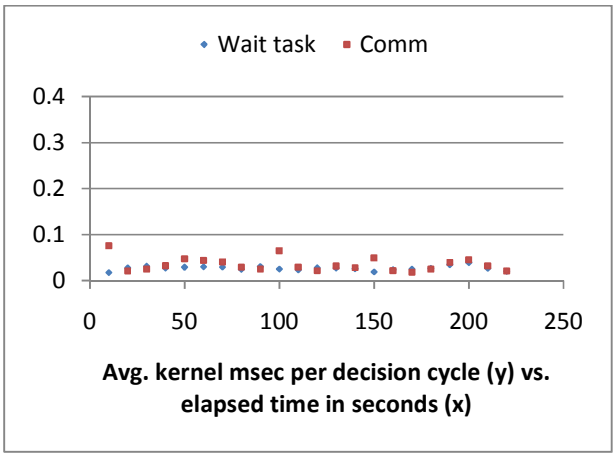

Figure 4. Kernel execution time results for the Comm agent.

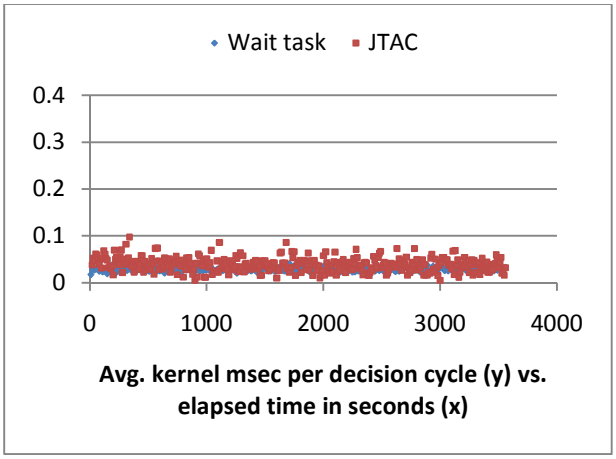

Figure 5. Kernel execution time results for the JTAC agent.

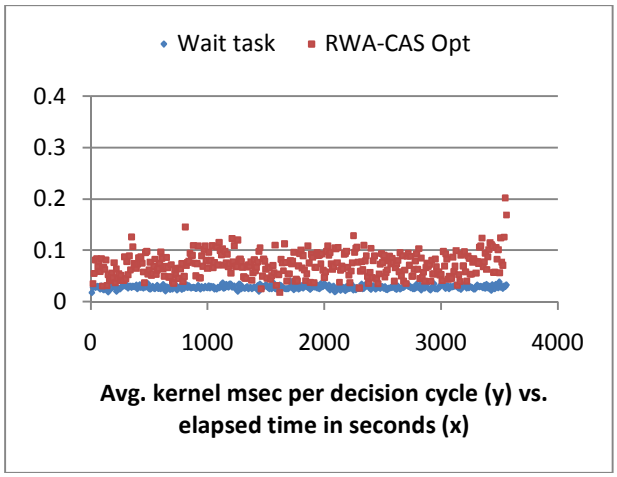

Figure 6. Kernel execution time results for the RWA-CAS agent.

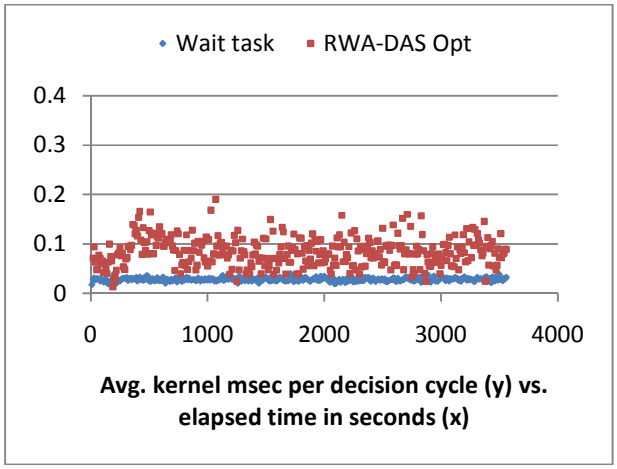

Figure 7. Kernel execution time results for the RWA-DAS agent.

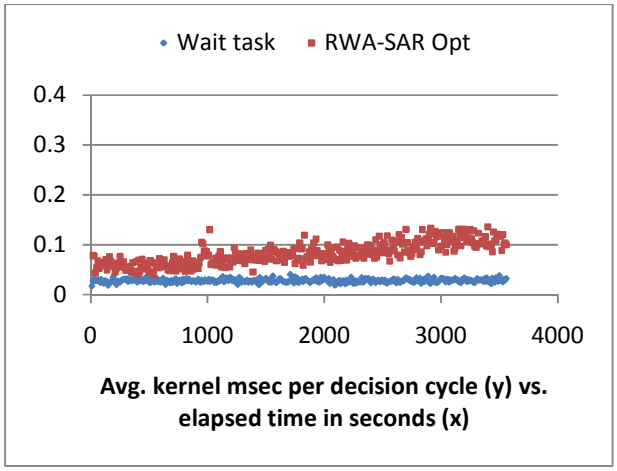

Figure 8. Kernel execution time results for the RWA-SAR agent.

Figure 4 through Figure 8 show the execution-time patterns over time for each agent in terms of time taken per decision. Recall that the "standard" response time we hope for from Soar-based systems is 50 milliseconds per decision. The first significant observation of these graphs is that run times are *far* faster than this typical target, being closer to the range of 100-200 *microseconds* per decision cycle. While average behavior is certainly of interest, it is also of interest to see where there are peaks in execution-time requirements. This is because such peaks represent the "least reactive" episodes that the agents engage in. While it may be fine, for example, for an agent to demonstrate average response times of 50 milliseconds, it could be the case that occasional episodes requiring 5-second response times (to be extreme) would make the agent basically worthless. Thus, we have plotted the data in scatter plots that summarize the data for each 10-second interval. From the figures, we can see that there is some variation in execution time within each agent, but the variation is within fairly narrow bands, with no egregious outliers. We can also see that different agents have different execution-time profiles, but in all cases the response times are extremely fast.

One of the major results here is that the agents (with one exception) do not slow down over time, showing that in general the applied agents show similar performance characteristics to the research agents that have been evaluated in the past. One exception is the RWA-SAR agent, which does appear to show a slight slowing trend over time. We have performed additional investigations and verified that there are not an increased number of production firings or working-memory changes over time, so the slowing must be coming from somewhere else. It is possible (perhaps even likely) that the slowing is coming from outside of the Soar process, but we will need to engineer Soar to collect new statistics in order to verify that. The current kernel time statistic does not measure "actual time in the kernel thread", so it can incur a penalty from external operating-system activities. In future experiments we will track down the precise nature of this slowing, as well as an explanation for why the RWA agent shows some slowing in the SAR scenario but not in the other RWA scenarios.

6. Practical Agent Performance

The previous experiments measured time spent in the Soar kernel. This was in part to omit any artificial degradation of performance, for example due to operating systems delays, etc. But in practical terms, what we care about for a deployed agent is its ability to react sufficiently quickly in real, wall-clock time. Thus, in addition to the kernel-time data, we collected statistics on the real wall-clock time per agent decision cycle in each experiment run. The average data per 10-second time slice appears in Figure 9, and the cumulative average appears in Figure 10. We include the cumulate average here because the graph is smoother and makes the overall trends and distinctions more clear. For the cumulative average, we have omitted the data collected during the first 10-second time slice, because the simulation environment has a comparatively expensive initialization time, relative to the agent cycle times. The results show that the two simplest agents (Wait task and Comm) have extremely low cycle times. The more complex agents appear to "settle" between 1.2 and 1.8 milliseconds per decision.

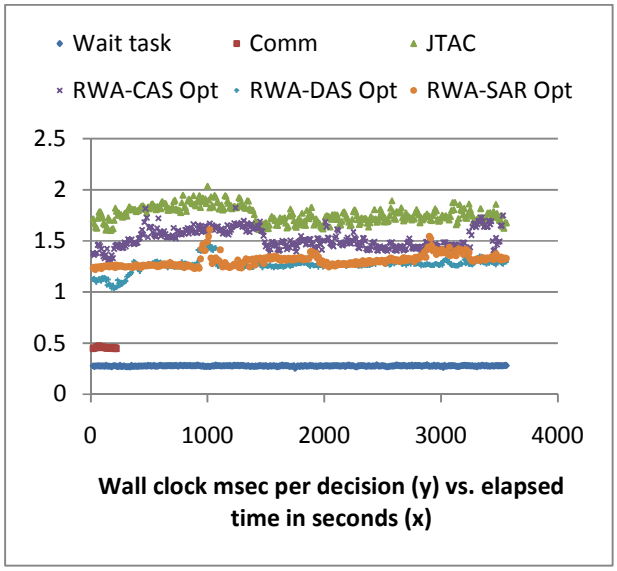

Figure 9. Real (wall-clock) execution time per agent decision.

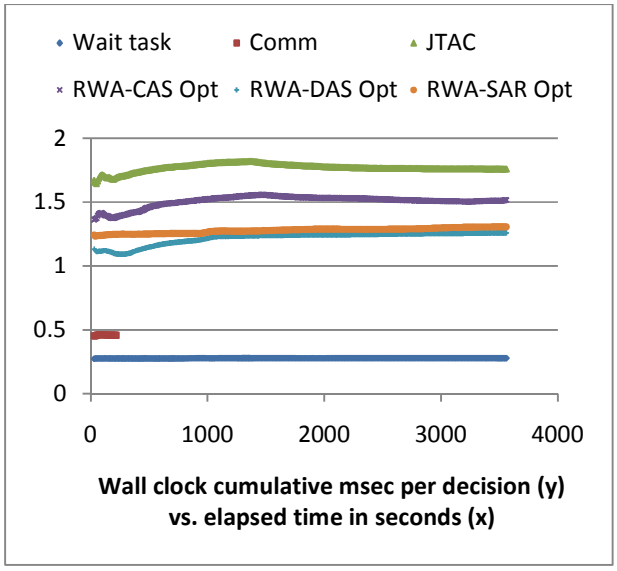

Figure 10. Cumulative average of real (wall-clock) execution time per agent decision.

Taken together, these results are extremely encouraging. Over an hour of run time, even the applied agent with the largest knowledge base (RWA) continues to run

efficiently. It is clear that different minimal levels of reaction time are suitable for different agent applications. For example, the Comm agent simulates communications processes that happen in the real world over the course of seconds, minute, and hours. Only the most dynamic tasks require reaction times at the millisecond level.

For a dynamic first-person shooter game, Laird and Duchi (2000) determined that humans subjectively judge 100 millisecond decision-cycle times in Soar agents to be the most "human looking" level of reactivity. If we use 50-100 milliseconds as a target range, then the results suggest that we could run in the neighborhood of fifty knowledge rich agents (or even more knowledge-lean agents) on a typical 2010 computation platform, with sufficient reactivity for human-level interaction. In applications in which an even coarser grain size of reactivity is appropriate, even more agents could run simultaneously on a single machine.

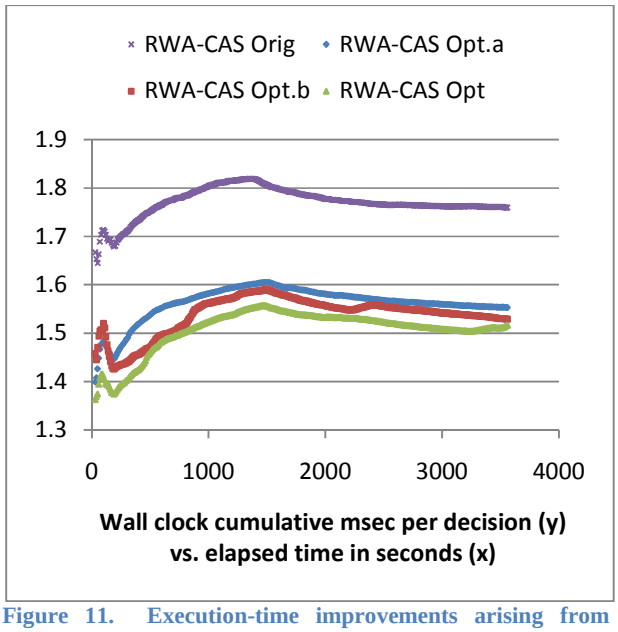

Figure 11. Execution-time improvements arising from iterative optimization of RWA code.

7. Opportunities for Improvement

As mentioned previously, all of the agents used in our experiments were taken "as is" from the SoarTech code respositories. Each agent was developed to achieve particular agent-capability goals and performance requirements, with run-time and memory efficiency addressed only to the extent necessary to meet the project requirements. However, in those cases where we do need to optimize performance, we have some particular "rules of thumb" to optimization approaches:

- Identify Soar productions in long-term memory that have very high firing rates, and find a way to reduce the firing rate. Strategies might include changing knowledge representations a bit, using "smarter" production representations, removing redundant conditions from productions, or moving rote computations to external functions. For example, it does not make much sense to have a Soar production continuously compute the range between two points. This is better done in a chunk of procedural code.
- Identify Soar productions that use high amounts of memory. This can arise from productions that have large numbers or combinations of partial matches in their conditions. In such cases, it can also be effective to adjust knowledge representations or possibly to split patterns into multiple productions in order to generate a more efficient implementation.

In the results presented above, we applied these basic rules of thumb in an attempt to find opportunities to improve the efficiency of the RWA-CAS agent. We made three rounds of optimizations, each involving finding a set of related "expensive" productions and then making the appropriate representation adjustments to reduce the expense without altering the behavior of the system. The results of these iterations are presented in Figure 11. Note that the Y axis has been "zoomed in" to make the differences between the four agents more apparent.

Each iteration of improvements led to a visible improvement in the real-time computational run-time of the RWA-CAS agent. There is a pattern of diminishing returns; each optimization resulted in less improvement than the previous optimization. However, with only a few relatively inexpensive optimizations, we were able to improve the performance of the RWA-CAS agent by about 14%. It also appears that additional rounds of optimization using this methodology (as opposed to a complete agent redesign) will not significantly improve the agent's performance.

In our analysis above, of "practical agent performance", we determined that the range of wall-clock decision-cycle time for knowledge-rich agents was in the neighborhood of 1.9 milliseconds. However, with only a few simple optimizations of the code, we were able to reduce that reaction time to just over 1.5 milliseconds. We can thus revise our estimate of the effective number of knowledge-rich, applied agents on a typical modern hardware platform to the neighborhood of five or six dozen agents.

We ran an additional multi-agent variation of the CAS experiment with 6 pairs of JTAC and RWA-CAS agents working together simultaneously. The results are shown in Figure 12. These results compare the performance of RWA-CAS in a 2-agent scenario (one JTAC and one RWA-CAS) to the performance of one of the RWA-CAS agents in a 12-agent scenario (six pairs of agents). These

results suggest that we cannot simply linearly extrapolate the performance statistics of a small number of agents to a larger number of agents. The real time per agent decision (averaged across all twelve agents) remains constant over time, but is higher than in the 2-agent case. This is another result that we plan to investigate further, with the hypothesis that the multi-agent overhead is external to the Soar process itself, perhaps in the operating system or the simulation environment.

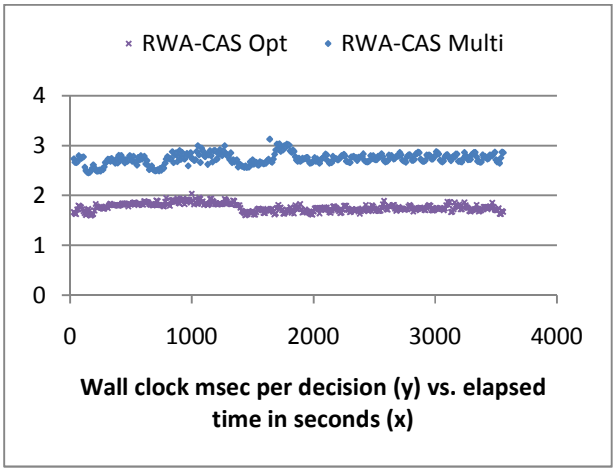

Figure 12. Comparison of real-time performance in a 2-agent scenario and a 12-agent scenario.

8. Conclusions

The experiments and analysis presented above provide evidence to support the following conclusions:

- Soar performance results demonstrated by research agents in other studies also carry over to applied agent systems of varying complexity that were not developed primarily as performance-evaluation systems. In general, applied Soar agents run much faster than required for human-like reaction times.
- The design of typical applied Soar agents does not imply performance degradation over the course of long-running tasks.
- There were opportunities to improve the performance of applied agents that were not designed with optimized performance as their primary goal but successive rounds of optimization soon stopped significantly impacting performance.
- The number of applied intelligent agents of fairly high complexity that can be expected to run within acceptable performance constraints on typical, modern computer hardware is somewhere in the dozens. However, there appears to be an as-yet-unknown overhead as the number of agents is multiplied.

These results should be encouraging to anybody considering the deployment of complex applied Soar agents. As was also demonstrated in previous studies on research agents, the performance of applied agents appear to depend more on the structure of those agents than on the complexity or run-time duration. Although Soar's underlying theory dictates cycle times of 50 milliseconds as a target, applied agents on typical modern hardware run at least an order of magnitude faster than that. Our experiments showed that there are some situations where we do not yet have full explanations for performance characteristics. This includes gradual slowing over time of the RWA agent when performing the SAR scenario, as well as an increased, potentially external, overhead when significantly multiplying the number of agents running simultaneously in a scenario. We plan to engineer new data-collection metrics into the Soar architecture to investigate these performance phenomena further.

9. References

Doorenbos, R. B. (1995). *Production Matching for Large Learning Systems*, Doctoral Dissertation, Computer Science Department, Carnegie Mellon Univ.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated Intelligent Agents for Combat Flight Simulation. *AI Magazine*, 20(1), 27-42.

Laird, J. E. (2009). *Millions of rules, billions of decisions*. Presentation at the 29th Soar Workshop.

Laird, J. E. (2008). Extending the Soar architecture. *Proceedings of the 2008 Conference on Artificial General Intelligence*.

Laird, J. E., & Duchi, J. C. (2000). Creating human-like synthetic characters with multiple skill levels: A case study using the Soar QuakeBot. *Proceedings of the AAAI Fall Symposium on Simulating Human Agents*.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987), Soar: An architecture for general intelligence. Artificial Intelligence, 33(3), 1-64.

Laird, J. E., Derbinsky, N., & Voigt, J. (2011). Peformance evaluation of declarative memory systems in Soar. *Proceedings of the 20th Behavior Representation in Modeling and Simulation Conference*.

Newell, A. (1990). *Unified theories of cognition.* Harvard University Press.

Taylor, G., Jones, R. M., Goldstein, M., Frederiksen, R., & Wray, R. E. (2002). VISTA: A generic toolkit for visualizing agent behavior. *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavior Representation.* Orlando, FL.

Taylor, G., & Ray, D. (2008). *Low fidelity tactical simulation environment: SimJr*. Presentation at the 28th Soar Workshop.